

Interfaces to tools:

applications, libraries,
Web applications, and **Web services**

Matúš Kalaš

The Bioinformatics Lab SS 2013

14th May 2013

Plurality of interfaces **for accessible bioinformatics:**

Applications

Web applications

Programmatic libs

Web services

interactive access
for humans

Applications

**Web
applications**

programmatic access
for applications

**Programmatic
libraries**

Web services

install and use locally

use over Web

4 things are most important to know about Web services:

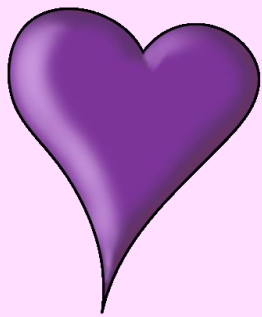
They provide programmatic access (API)

They are accessible over the Web

that means over HTTP

They can be in 2 flavours: SOAP and REST

And ...



Web services



are lovely!



Exercises:

Calling a Web service

from SoapUI and programmatically

Developing a Web service interface

writing your own WSDL file

Testing a mock-up of your service

from SoapUI and programmatically

Calling a Web service manually:

Try for example the JasparDB Web service (it's a SOAP Web service)

`http://cbu.bioinfo.no/wsdl/JasparDB.wsdl`

Install SoapUI

SoapUI is great, but it only supports WSDL 1.1 at the moment

Call the Web service from the SoapUI

1. File -> New soapUI project
2. Initial WSDL/WADL: `http://cbu.bioinfo.no/wsdl/JasparDB.wsdl`
3. Make sure Create Requests is ticked on, click OK
4. Open JasparDB -> getMatrixById -> Request 1
5. Change the content of <jas:Id> to e.g. MA0062, click the green arrow and wait
6. Say hooray, you've got the result!

Calling a Web service programmatically:

Python with SUDS library

```
>>> from suds.client import Client
>>> client = Client('http://cbu.bioinfo.no/wsdl/JasparDB.wsdl')
>>> print client.service.getMatrixById('MA0062').A.col[0]
```

Java with Axis2 library and ADB data binding. First generate a service proxy using Axis2

```
package org.example.test.jaspar;

import java.rmi.RemoteException;
import no.uib.bccs.jaspar.JasparDBStub;
import no.uib.bccs.jaspar.JasparDBStub.Database_type1;
import no.uib.bccs.jaspar.JasparDBStub.Format_type1;
import no.uib.bccs.jaspar.JasparDBStub.GetMatrixById;

public class TestJaspar
{

    public static void main(String[] args) throws RemoteException
    {
        GetMatrixById input = new GetMatrixById();
        input.setId("MA0062");
        input.setDatabase(Database_type1.CORE);
        input.setFormat(Format_type1.PFM);

        JasparDBStub jaspar = new JasparDBStub();
        System.out.println(jaspar.getMatrixById(input).getMatrix().getA().getCol()[0]);

    }

}
```

Developing a Web service interface:

Now define your own Web service

Use an XML editor of choice:

XML Copy Editor (free)

oXygen XML Editor (free trial)

XMLSpy (free trial; best but Windows only)

vim or emacs

eclipse

Look at the FreeContact WSDLs:

http://rostlab.org/~lkajan/freecontact_soap.wsdl

SOAP & WSDL 1.1

http://rostlab.org/~lkajan/freecontact_rest.wsdl

REST & WSDL 2.0

Try the same with your tool, validate the WSDL

Testing a mock-up of a Web service:

In SoapUI

1. File -> New soapUI project
2. Initial WSDL/WADL:
 - a) first with *http://roslab.org/~lkajan/freecontact_soap.wsdl*
 - b) then the WSDL you've created
3. Make sure Create Requests is ticked on
4. Tick Create MockService, click OK
5. Tick Start MockService, check that Add Endpoint is ticked too. Click OK
6. Now you can edit the responses of the mock-up server if you wish

Test your mock-up service manually from soapUI

Don't forget to choose the mocked-up endpoint URL

Test your mock-up service programmatically from your favourite language

Don't forget to first set the endpoint in the WSDL to the mocked-up endpoint URL!

Advanced:

Implement your Web service

1. Generate code stubs for your favourite language from your WSDL
2. Fill in some business logic
3. Deploy onto your local Web service environment / Web server
4. Test from SoapUI (now don't send requests to the mocked-up endpoint but to the deployed one)
5. Test, test, test, from various programming languages (python SUDS is the simplest to try 1st)

Implement an asynchronous Web service

with queue for scheduling jobs in the backend

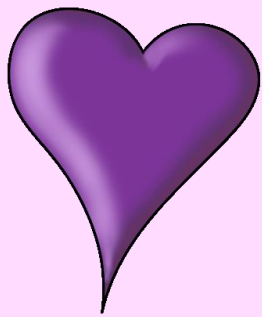
with operations such as: submitJob, getStatus, getResult

(Don't yet😊) Implement a Web service exchanging massive data (100MB to TB)

Don't try this at home! 😊 Or rather, not before you're confident with Web services...

It's possible, however. Standard option for *slightly-massive* data is MTOM.

Easiest option for *semi-massive* data is to send a URL of the data (http or ftp).



Web services



are lovely!

